Bluetooth Adaptive Frequency Hopping and Scheduling

N. Golmie, O. Rebala, N. Chevrollier
National Institute of Standards and Technology
Gaithersburg, Maryland 20899
Email: nada.golmie@nist.gov

Abstract— **In this paper, we investigate the use of Adaptive Frequency Hopping (AFH) techniques aimed at modifying the Bluetooth frequency hopping sequence in the presence of WLAN direct sequence spread spectrum devices. We examine the conditions such as the applications, topologies, and scenarios under which AFH techniques improve performance that is measured in terms of packet loss, TCP delay, and channel efficiency. We also compare the results obtained with AFH to others obtained using a scheduling technique that consist in delaying the transmission of a Bluetooth packet until the medium is "idle". Our results show that an obvious performance improvement with AFH is in terms of delay and throughput. AFH brings the delay down to the same level than when no interference is present. On the other hand, AFH is rather slow in responding to changes in the environment and the packet loss is more significant than with the scheduling. This is probably due to the limitations imposed by the communication overhead. The main difficulty for AFH is having to dynamically communicate the changes to all slaves in the piconet in order to keep the synchronization.**

I. Introduction

The deployment of different wireless devices for mobile, home, and enterprise networks, all operating in the 2.4 GHz unlicensed band, is met with growing concerns about signal interference and performance degradation. To address these challenges, a number of industry led activities have focused on the coexistence of these devices in the same environment. For example, the IEEE 802.15.2 Coexistence Task Group and the Bluetooth Special Interest Group (SIG) are looking at techniques for alleviating the impact of interference between IEEE 802.11b and Bluetooth devices.

A solution that has gained acceptance in both groups is based on modifying the frequency hopping sequence of Bluetooth in order to make it avoid direct sequence spread spectrum devices such as IEEE 802.11b. This so-called Adaptive Frequency Hopping (AFH) has gained momentum especially after the Federal Communications Commission, a US government agency in charge of telecommunication regulations, has relaxed the minimum frequency hop requirement to 15 (down from 75). AFH is expected to be included in the new release of the Bluetooth specifications, Version 1.2.

Other proposals considered by the groups range from collaborative schemes intended for Bluetooth and IEEE 802.11 protocols to be implemented in the same device to fully independent solutions that rely on interference detection and estimation. Except for a Time Division Multiple Access (TDMA) technique aimed at time sharing the Bluetooth and 802.11 signals [1], most mechanisms considered do not require any direct communication between the protocols. For example, Bluetooth Interference Aware Scheduling (BIAS) is a MAC scheduling technique [2] that is aimed at delaying packet transmission if the medium is used by other devices. Another technique known as OverLap Avoidance (OLA) [3] uses different Bluetooth encapsulations to avoid a frequency collision between Bluetooth and 802.11.

Our goals in this paper are to investigate the use of AFH techniques aimed at modifying the Bluetooth frequency hoping sequence in the presence of WLAN direct sequence spread spectrum devices. Mainly, under what conditions – i.e., interference levels, topologies, scenarios, and applications – is it practical to use either AFH or BIAS? Which mechanisms is more effective for a given application? How fast can either technique adjust to changes in the environment? We conduct numerous simulation experiments to evaluate and quantify the operation range of AFH and BIAS. To answer the question of application sensitivity, we consider four applications, namely, voice, video, HTTP, and FTP. We set the application profiles available in the OPNET library including the details of the entire TCP/IP stack.

In section II, we describe an AFH algorithm implementation. In section III, we describe BIAS. Section IV discusses channel estimation techniques and their use with interference mitigation schemes. In section V, we consider several experiments to evaluate the performance of AFH and how it compares to BIAS. In section VI, we offer concluding remarks.

II. Bluetooth Adaptive Frequency Hopping

We devise an AFH algorithm that modifies the original Bluetooth frequency hopping scheme as follows.

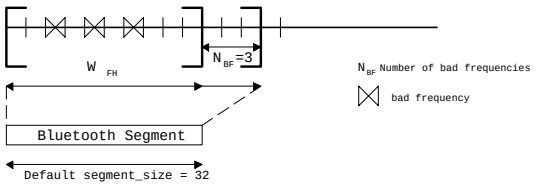

Fig. 1. Resizing the Frequency Hopping Window, W_{FH}

Given a sorted list of odd and even frequencies, and a segment of 32 frequencies, $W_{FH} = 32$, including "good" and "bad" frequencies, the algorithm visits each "good" frequency exactly once. While the segment size is the same as the one used in the current Bluetooth specifications [4], in order to filter out the so-called "bad" frequencies, the window, W_{FH}, over which frequencies are selected is increased by the number of "bad" frequencies, N_{BF}, in W_{FH}. Thus, the main difference between the scheme we propose and the current Bluetooth specifications

is the resizing of the interval over which frequencies are randomly selected for each segment as illustrated in Figure 1.

Note that in order for a frequency to be classified "bad", it has to be "bad" for at least one device in the piconet. Thus, N_{BF} represents the union of the sets of "bad" frequencies collected from of all devices.

1: $W_{FH} = segment\_size$; // Initialize the hopping algorithm window size
2: $W_{FH} += N_{BF}$; // Increase by the number of "bad" frequencies
3: If ($W_{FH} > 79$)
4: $W_{FH} = 79$; // limit to the list size
5: $N_{BF} = min(N_{BF}, 79 - H_{min})$
6: //use at least H_{min} different frequencies

After, each "good" frequency is visited once, a new segment is set including 16 frequencies of the previous segment and 16 new frequencies in the sorted list.

When W_{FH} is greater than 79, the number of "good" frequencies may be less than 32 and therefore there are not enough "good" frequencies to fill in the segment. In that case, we allow each "good" frequency to be visited more than once, with the condition to use at least H_{min} different frequencies. In other words, we impose the minimum hop set to be at least equal to H_{min} different frequencies. In our simulations, we set $H_{min} = 15$.

In summary, the difference between AFH and the original Bluetooth hopping sequence algorithm is the dynamic resizing of W_{FH} based on the frequency classification status. The other requirement for AFH is the exchange of LMP messages between the master and the slaves in the piconet in order to advertise the new hopping sequence.

Finally, it is worth pointing out that the details presented here give an example of how "bad" frequencies can be eliminated from the Bluetooth hopping sequence. Other variants are also possible. For example, the IEEE 802.15.2 Task Group on coexistence considers a more general algorithm that allows one to choose which "bad" frequencies to keep and which to eliminate. However, for all practical scenarios considered, most AFH algorithms will give comparable performance. In fact, this is easily verified by implementing the AFH in [5], denoted by *AFH-IEEE*, and comparing the results obtained to the algorithm proposed in this paper.

III. Bluetooth Interference Aware Scheduling

The Bluetooth Interference Aware Scheduling (BIAS) algorithm [6] is a delay policy implemented at the master device that postpones the transmission of a packet until a slot associated with a "good" frequency becomes available. The master device, which controls all data transmissions in the piconet, uses information about the state of the channel in order to avoid data transmission to a slave experiencing a "bad" frequency. Furthermore, since a slave transmission always follows a master transmission, using the same principle, the master avoids receiving data on a "bad" frequency, by avoiding a transmission on a frequency preceding a "bad" one in the hopping pattern.

This simple scheduling scheme needs only be implemented in the master device and translates into the following transmission rule. *The master transmits in a slot after it verifies that both the slave's receiving frequency and its own receiving frequency are "good". Otherwise, the master skips the current transmission slot and repeats the procedure over again in the next transmission opportunity.*

Additional considerations including bandwidth requirements and quality of service guarantees for each master/slave connection in the piconet can also be combined with the channel state information and mapped into transmission priorities given to each direction in the master/slave communication. Details on assigning transmission priorities are given in [6].

The algorithm's general steps are summarized below.

1: Every Even TS_f // Master transmits on frequency f
2: if $TS_f + l_{dn}$ is good // Master can receive in next slot
3: {
4: A^f_{data} = {set of slaves s.t. ((f "good") and ($qsize > 0$) }
5: if ($A^f_{data} \neq \emptyset$)
6: select slave i //according to a priority criteria
7: transmit data packet of size l_{dn} to slave i
8: }

where l_{dn} is the length of the packet from the master to the slave and TS_f is the transmission slot using frequency f.

IV. Channel Estimation

Channel estimation methods include BER calculation, packet loss, or frame error rate measurements performed by each receiver (master and slave device). Since in a Bluetooth piconet, the master device controls all packet transmission, the measurements collected by the slave devices are sent to the master that decides to (1) either avoid data transmission to a slave experiencing a "bad" frequency, and/or (2) modify the frequency hopping pattern. While in the former case the decision remains local to the master, in the latter case, the master needs to communicate the change to all slaves in the piconet in order to maintain synchronization. Also, the former method falls into the scheduling policy category, while the latter is in the AFH category.

Channel estimation is based on measurements conducted on each frequency in order to determine the presence of interference. Although our discussion exclusively focuses on packet loss, other measurements can be used. In a nutshell, channel estimation works as follows. Each Bluetooth receiver maintains a *Frequency Status Table* (FST) where a percentage of packets dropped due to errors, Pr(PLoss), is associated to each frequency offset, f, as shown in Figure 2. Frequencies are classified "good" or "bad" depending on whether their packet loss rate is below or above a threshold value respectively. In Figure 2 the threshold value is equal to 0.5. Each slave has its own FST maintained locally. However, the master has in addition to its FST, a copy of each slave's FST.

At regular time intervals each slave updates its FST copy kept at the master using a status update message that can be defined in the Layer Management Protocol (LMP). Alternatively, the master can derive information about each slave's FST by keeping track of the ACK bit sent in the slave's response packet.

First, we define two phases in the channel estimate procedure. During the *Estimation Window*, EW, packets are sent on all frequencies regardless of their classification. EW is followed

Status	Frequency Offset	Pr[PLoss]
good	0	10^{-3}
bad	1	0.75
bad	2	1
bad	3	0.89
	...	
good	76	10^{-4}
good	77	10^{-3}
good	78	10^{-3}

Fig. 2. Frequency Status Table

by an interval, EI, in which slaves have updated their FST at the master (refer to Figure 3). The master uses the channel information collected during EW in order rearrange the frequency hopping pattern in case of AFH and/or selectively avoid to transmit packets on so-called "bad" frequencies. In order to avoid having to manually compute or pick an arbitrary value for EW, we use a technique to dynamically adjusts the window based on the number of times, N_f, each frequency in the band should be visited. Further details on channel estimation parameter tuning are available in [6]. In our simulations, we use $N_f = 1$, $EI_{min} = 2$s, $EI_{max} = 100$s.

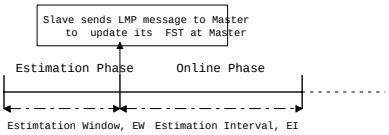

Fig. 3. Explicit Estimation

Note that during both phases, Pr(PLoss) is measured and continuously updated. Although the local FSTs can be updated every time a packet is received, the copy of the slave FST kept at the master is updated either at the end of each EW using an LMP defined message, or every time a packet acknowledgement (ACK) is received by the master. It is important to point out that for scheduling purposes, the master can make use of the ACK feedback information as soon as it becomes available. On the other hand, AFH requires a master to slave message exchange in order to keep the piconet synchronized. In our study, we assume that updates are based on ACK feedback for BIAS and LMP messages for AFH sent at the end of each EW.

V. Performance Evaluation

In this section we present simulation results to evaluate the performance of AFH in a realistic environment. We ran several experiments using different applications, and network topologies. We consider four application profiles, namely, FTP, HTTP, voice, and video. We use the TCP(UDP)/IP stack implemented in the OPNET library and configure the application parameters provided. For the FTP profile, the parameters are the percentage of put/get, the inter-request time, and the file size. The percentage of put/get represents the number of times the put command is executed in an FTP connection over the total number of put and get commands, i.e., a fifty percent indicates that half of the FTP commands executed are put, and the other half are get. The inter-request time is the interval between two FTP commands, and the file size represents the size of the file requested in bytes. The HTTP profile is described by parameters characterizing a web page such as the page interarrival time, the number of objects in each page and their size in bytes. For the voice application, we use the encoding defined in the G.723.1 specifications. The video application uses a 1 Frame/s rate and a frame size of 17280 bytes. The application profile parameters are summarized in Table I.

TABLE I
Application Profile Parameters

Parameters	Distribution	Value
Bluetooth FTP		
Percentage of Put/Get		100%
Inter-Request Time (seconds)	Exponential	5
File Size (bytes)	Constant	2M
Bluetooth HTTP		
Page Interarrival Time (seconds)	Exponential	5
Number of Objects per page	Constant	2
Object 1 Size (bytes)	Constant	10K
Object 2 Size (bytes)	Uniform	(20K,600K)
Bluetooth Voice		
Encoder		G.723.1
Bluetooth Video		
Frame Rate	Constant	1 Frame/s
Frame Size (bytes)	Constant	17280 (128 x 120 pixels)
WLAN FTP		
Percentage of Put/Get		0%
Inter-Request Time (seconds)	Exponential	1
File Size (bytes)	Constant	2M
Connection Duration (seconds)	Exponential	20
Interval between Connections (seconds)	Exponential	20

For each network topology considered, we run a set of 16 simulations covering each application and algorithm combination. *None* refers to the case when no algorithm is present, while *BIAS* and *AFH* refer to using BIAS and AFH respectively. Note that *AFH-IEEE* refers to the AFH algorithm included in the draft IEEE Recommended Practice on Coexistence [5]. Performance is measured in terms of the packet loss, the delay measured at the TCP layer (in seconds), and the channel efficiency. The channel efficieny measures the normalized number of data packets received minus the number of packets lost and packets ignored in the case of duplicate transmissions. Averages are obtained and reported for each simulation set consisting of 10 simulation runs. Each simulation is run for 900 seconds. The packet loss and channel efficiency are measured at the application client (master device), while the TCP access delay is measured at the application server (slave device).

A. Experiment 1: WLAN Interference

We use Topology 1 illustrated in Figure 4 with one WLAN system (Access Point/Station) and a Bluetooth master/slave pair. The WLAN access point (AP) is located at (0,15) meters, and the WLAN station is fixed at (0,1) meters. The Bluetooth slave device is fixed at (0,0) meters and the master is fixed at (1,0) meters.

In this case, the WLAN station is "uploading" files to WLAN server using the FTP *put* command. A summary of the application profile is described in Table I.

Table II gives the performance of the Bluetooth FTP application. First, observe that the results with AFH and AFH-IEEE are comparable and therefore in our discussion we will not distinguish between the two algorithms unless specified otherwise.

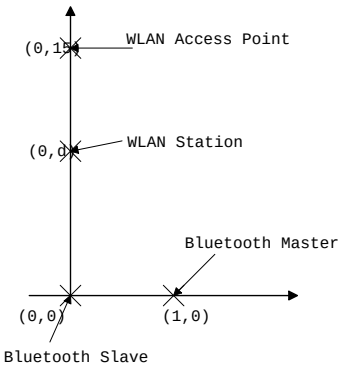

Fig. 4. Topology 1 - Two WLAN devices and one Bluetooth piconet

TABLE II
EXPERIMENT 1: BLUETOOTH FTP PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.1633	0.0009	0.0748	0.0721
TCP Delay (seconds)	0.0201	0.0178	0.0167	0.0184
Channel Efficiency	0.6921	0.9981	0.9306	0.9336

When no interference mitigation algorithm is present, which represents a base case, the packet loss is around 16%. The effects of BIAS are summarized in comparison to the base case as follows. First, we observe a decrease in packet loss to negligible levels, a decrease of 3 ms in the delay (from 20.1 to 17.8 ms), and an increase of 30% in the efficiency. Similarly the effects of AFH are characterized by a lower packet loss (to 7%), lower delay (16.7$\sim$ 18.4ms), and higher efficiency ($\sim$93%). The delays observed with BIAS and AFH are almost comparable, while the difference in efficiency is more striking. Although more packets are sent with AFH, they are more likely due to duplicate transmissions.

The observations noted for FTP are also consistent with the HTTP results given in Table III. Similarly, BIAS reduces the packet loss to zero, the access delay by 6 ms (to 11 ms), and increases the efficiency by 30% (to 99%). On the other hand, AFH gives a packet loss of 5%, reduces the delay by 3 ms (to $\sim$ 10 ms) and increases efficiency by 20% (to $\sim$ 95%).

The results of the video application are shown in Table IV. Here again, BIAS reduces the packet loss to a negligible level,

TABLE III
EXPERIMENT 1: BLUETOOTH HTTP PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.1487	0.0012	0.0585	0.0445
TCP Delay (seconds)	0.0171	0.0112	0.0109	0.0107
Channel Efficiency	0.6943	0.9976	0.9453	0.9557

TABLE IV
EXPERIMENT 1: BLUETOOTH VIDEO PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.1310	0.0043	0.0455	0.0269
Channel Efficiency	0.6974	0.9914	0.9503	0.9611

TABLE V
EXPERIMENT 1: BLUETOOTH VOICE PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.1359	0.0091	0.0400	0.0212
Channel Efficiency	0.6901	0.9840	0.9631	0.9722

and increases the efficiency to 99%. On the other hand, AFH causes a decrease in packet loss to 4.5% and 2.6% for AFH and AFH-IEEE respectively (down from 13%).

Table V shows the results of the voice application. We observe a packet loss of 4 and 2% with AFH and AFH-IEEE respectively compared to 0.9% with BIAS. The channel efficiency is 98%, 96%, and 97% for BIAS, AFH, and AFH-IEEE respectively.

For AFH, the time it takes to estimate the channel and communicate the changes is usually longer than for BIAS leading to a higher packet loss and a lower channel efficiency. This signifies that a number of packets transmitted are due to duplicate transmissions that end up getting discarded at the destination and therefore do not lead to a higher goodput. This observation captures the essence of the performance trade-offs between AFH and BIAS. AFH increases the total number of packets sent at the cost of higher packet loss, and lower efficiency. This may be acceptable for some bandwidth hungry applications such as FTP and HTTP, but perhaps less desirable for real-time applications such as voice and video. In summary, there are definite trade-offs for using AFH versus BIAS depending on the application considered.

B. Experiment 2: Multi-WLAN Interference

In this experiment, our goal is to study the performance of AFH in a multi-WLAN environment, where the Bluetooth hopping sequence is further reduced. We use Topology 2 illustrated in Figure 5, consisting of 2 WLAN systems (source-sink pairs) operating on non-overlapping frequencies (each WLAN system operates on a different center channel). We use the same traffic parameters described in Table I.

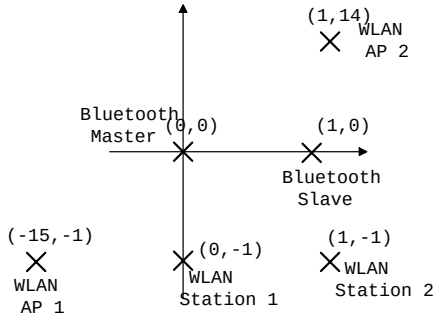

Fig. 5. Topology 2 - Multi-WLANs and Bluetooth piconets interference

Since there are two WLAN systems occupying about 16 frequencies each, that leaves about 47 frequencies in the band to be used by Bluetooth. With BIAS, the Bluetooth piconet only transmits on "good" frequencies, and therefore has to skip approximately 1 in every 3 transmission opportunities. With AFH,

TABLE VI

EXPERIMENT 2: BLUETOOTH FTP PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.3431	0.0183	0.1524	0.1542
TCP Delay (seconds)	0.0322	0.0213	0.0218	0.0242
Channel Efficiency	0.4500	0.9684	0.8486	0.8552

TABLE VII

EXPERIMENT 2: BLUETOOTH HTTP PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.2535	0.0169	0.1350	0.1172
TCP Delay (seconds)	0.0181	0.0191	0.0160	0.0152
Channel Efficiency	0.4725	0.9705	0.8668	0.8849

the frequency hopping sequence is modified in order to include only "good" frequencies. Therefore, one expects significant throughput and delay improvements with AFH. Our goals in this experiment are to verify that our previous conclusions about AFH and BIAS still hold even in the case of severe interference.

Table VI gives the performance results for the Bluetooth FTP application. The packet loss when no algorithm is present is around 34% for Bluetooth. Note that it is more than double the packet loss obtained in Experiment 1. The packet loss is 1.8%, 15.24%, 15.42% with BIAS, AFH, and AFH-IEEE respectively. Delays with AFH and BIAS are comparable (21 ms). On the other hand, the channel efficiency is only 84% and 85% with AFH, while it is around 96% with BIAS.

Table VII gives the results for the Bluetooth HTTP application. The results are consistent with the FTP results for the most part. There are additional delay improvements with AFH.

Tables VIII and IX give the results for the video and voice applications respectively. The general trends observed in Experiment 1 are still valid. In general, BIAS leads to lower packet loss and higher or equal channel efficiency than AFH.

VI. CONCLUDING REMARKS

In this paper, we study using adaptive frequency hopping for Bluetooth devices when operating in close proximity to WLAN systems. We present the details of an AFH algorithm and compare its performance to BIAS, a delay transmission method aimed at interference mitigation.

A summary of our findings is as follows. For the applications considered, BIAS leads to a lower packet loss and an equal or higher channel efficiency than AFH. Basically, when the channel estimation has to be performed often, the synchronization overhead associated with AFH leads to an additional packet loss. In fact, our results indicate that this packet loss is often accompagnied with additional duplicate packet transmissions, which in turn lead to a lower channel efficiency. Thus, the number of additional packets transmitted with AFH is often offset by an additional number of packets lost or ignored. In other words, the adaptive part of AFH is constrained by the channel estimation and how often to synchronize the devices in the piconet. That in turn determines the response time and the performance.

Having said that, AFH may be more suitable for slow-changing environments where the same sequence could be used for a long period of time. On the other hand, in environments where the interference levels vary more rapidly, BIAS would be the interference mitigation solution of choice.

An area of future investigations would be combining BIAS and AFH within the same scenario, where BIAS would be used to respond quickly to a change in the environment, before an AFH policy is put in place if the interference persists for a long period of time.

TABLE VIII

EXPERIMENT 2: BLUETOOTH VIDEO PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.2725	0.0230	0.1070	0.0750
Channel Efficiency	0.2079	0.9803	0.8485	0.8878

TABLE IX

EXPERIMENT 2: BLUETOOTH VOICE PERFORMANCE

	None	BIAS	AFH	AFH-IEEE
Packet Loss	0.2126	0.0433	0.0940	0.0564
Channel Efficiency	0.4543	0.9269	0.9088	0.9300

REFERENCES

[1] J. Lansford, A. Stephens, and R. Nevo, "Wi-Fi (802.11b) and Bluetooth: Enabling Coexistence," in *IEEE Network Magazine*, Sept/Oct. 2001, vol. 15, pp. 20–27.

[2] N. Golmie, "Interference Aware Bluetooth Scheduling Techniques," in *IEEE P802.11 Working Group Contribution, IEEE P802.15-01/143r0*, Hilton Head, NC, March 2001.

[3] Carla F. Chiasserini, and Ramesh R. Rao, " Coexistence mechanisms for interference mitigation between IEEE 802.11 WLANs and bluetooth ," in *Proceedings of INFOCOM 2002*, 2002, pp. 590–598.

[4] Bluetooth Special Interest Group, "Specifications of the Bluetooth System, vol. 1, v.1.0B 'Core' and vol. 2 v1.0B 'Profiles'," December 1999.

[5] IEEE Std. 802-15 Task Group on Coexistence, "Draft Recommended Practice for Information Technology, Part 15.2: Coexistence of Wireless Personal Area Networks with Other Wireless Devices Operating in the Unlicensed Frequency Bands," March 2003.

[6] N. Golmie, "Bluetooth Dynamic Scheduling and Interference Mitigation," in *ACM Mobile Network, MONET*, 2002.